RECEIVED
CENTRAL FAX CENTER
NOV 2 3 2004

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:
Broussard

Serial No. 09/870,621

Filed: May 31, 2001

For: COMBINING THE FUNCTIONALITY OF MULTIPLE TEXT CONTROLS IN A GRAPHICAL USER INTERFACE

Group Art Unit: 2173
Examiner: Bonshock, D.

Atty. Dkt. No. AUS920010263US1
(5468-07600)

I hereby certify that this correspondence is being transmitted via facsimile or deposited with the U.S. Postal Service with sufficient postage as First Class Mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313, on the date indicated below:

11/23/2004
Date

Kevin L. Daffer

APPEAL BRIEF

Box AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313

Sir:

Further to the Notice of Appeal faxed to and received by the U.S. Patent and Trademark Office on September 24, 2004, Appellant presents this Appeal Brief. The Notice of Appeal was filed following mailing of a final Office Action on June 30, 2004. Appellant hereby appeals to the Board of Patent Appeals and Interferences from a final rejection of claims 1-25 in the final Office Action, and respectfully requests that this appeal be considered by the Board.

I. REAL PARTY IN INTEREST

The subject application is owned by International Business Machines Corporation, a corporation having its principal place of business at New Orchard Road, Armonk, New York, 10504, as evidenced by the assignment recorded at Reel 011888, Frame 0379.

II. RELATED APPEALS AND INTERFERENCES

Notices of Appeal have been filed for the following applications, which share a common specification with the application currently on appeal.

09/870,622 Notice of Appeal filed 8/24/04

09/870,615 Notice of Appeal filed 9/14/04

However, because dissimilar art is cited in the present application and the above-mentioned related applications, Appellants do not believe that the outcome of this appeal will have any bearing on the Board's decision on the related appeals. No other appeals or interferences are known which would directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

III. STATUS OF CLAIMS

Claims 1-25 were originally filed in the present application. Claims 1, 3-6, 8, 10, 11, 13, 15-18, 20, 21, 23, and 25 were amended in a response filed April 15, 2004 to an Office Action mailed January 15, 2004. Claims 1-25 stand finally rejected under 35 U.S.C. § 103, which are the subject of this appeal. A copy of claims 1-25, as on appeal (incorporating entered amendments), is included in the Appendix hereto.

IV. STATUS OF AMENDMENTS

No amendments to the claims were filed subsequent to their final rejection. The Appendix hereto therefore reflects the current state of the claims.

V. SUMMARY OF THE INVENTION

Appellant's claimed invention relates to a system of software components, a computer-readable storage device and a method for dynamically switching between two proxy components, depending on a mode of use of an object displayed in a graphical user interface (GUI).

In some embodiments, the system of software components (172, 174 and 176, Fig. 17) is configured for displaying an object (170, Fig. 17) created by an application program (APP, Fig. 1) running under an operating system (OS). In particular, the system of software components may include a first proxy component (e.g., JTextFieldProxy 174, Fig. 17), a second proxy component (e.g., JPasswordFieldProxy 176), and a peer component (JTextFieldPeer 172). The peer component may be used for selecting either the first proxy component or the second proxy component, depending on a mode of use of the object. In some cases, for example, the selection between proxy components may depend on the status of an echo character, which specifies whether entered text should be displayed as masked (e.g., "*****") or unmasked (e.g., "hello") text. In most cases, the selection can be made during runtime. After the proxy component is selected, however, the selected proxy component functions to dynamically create a new graphics resource component (e.g., a new text field or password field control) for displaying the object. It is important that the appearance of the displayed object be substantially independent of the operating system. In this manner, the presently claimed system of software components provides a unique mode-switching capability, which may permit two Swing components (e.g., the JTextField and JPasswordField components) to alternate as replacements for an AWT TextField component, depending on the manner in which the AWT TextField component is being used by the application. (Specification -- page 38, line 17 to page 39, line 25).

In some embodiments, the method (Fig. 18) is configured for displaying an object (170, Fig. 17) created by an application program (APP, Fig. 1) running under an operating system (OS). In particular, the method may use at least one of a system of software components, which can be invoked during runtime to generate a graphical representation of the object. As noted above, the graphical representation of the object is substantially independent of the operating system. The method, which may be performed by the application program, may include: activating a first proxy component (e.g., JTextFieldProxy 174, Fig. 17) of the system of software components to dynamically generate a first graphical representation of the object (e.g., a Text Field box) during runtime. Next, the method may monitor the mode of use of the object. For example, the method may monitor the status of an echo character to determine if there has been a change in status (178, Fig. 18). Upon detecting a change in the mode of use of the object, the method may deactivate (180, 182) the first proxy component and activate (184, 186, 188) a second proxy component (e.g., JPasswordFieldProxy 176, Fig. 17) of the system of software components to dynamically generate a second graphical representation of the object (e.g., a Password Field box) during runtime. The second graphical representation may be distinct from the first graphical representation; however, each may be substantially independent of the operating system. In this manner, the method

described herein may select either the first or the second proxy component, depending on a status of a software flag (e.g., an echo character indication) associated with the object. The object may then respond to text entry events in a manner corresponding to the status of the software flag. (Specification -- page 39, line 27 to page 40, line 6).

In some embodiments, the computer-readable storage device (18, Fig. 1) may include a windows-based operating system (OS, Fig. 1), an application program (APP, Fig. 1) running under the operating system, and an object (170, Fig. 17) created at runtime by the application program and adapted for multiple modes of use by the application program. In a particular embodiment, the "multiple modes of use" may correspond to the use of a Text Field as either a normal Text Field, which displays unmasked text, or a Password Text Field, which displays masked text. In any case, the application program may be adapted for performing the method described above and shown in Figs. 17 and 18. As such, the application program may be adapted for: activating a first proxy component to dynamically generate a first graphical representation of the object during runtime; monitoring the mode of use of the object; and upon detecting a change in the mode of use of the object, deactivating the first proxy component and activating a second proxy component to dynamically generate a second graphical representation of the object during runtime. As noted above, the second graphical representation may be distinct from the first graphical representation; however, each may be substantially independent of the operating system running the application.

VI. ISSUES

1. Whether claims 1-25 are unpatentable under 35 U.S.C. §103(a) over WinZip computing Inc., WINZIP 8.0 (hereinafter "WinZip"), in view of the publication referred to as Java Platform 1.2 Beta 4 API Specification: Class JPasswordField and Class JTextField (hereinafter "Java"), and in further view of U.S. Patent No. 5,327,529 to Fults et al. (hereinafter "Fults").

VII. GROUPING OF CLAIMS

Claims 1-12 (Group I) stand or fall together.
Claims 13-25 (Group I) stand or fall together.

Reasons as to why the two group of claims are believed to be separately patentable are explained below in the appropriate parts of the Argument.

VIII. ARGUMENT

In general, the specification provides a unique system of software components that enables an application program (e.g., a Java application) to be truly portable across all operating system (OS) platforms by providing various means for maintaining the "look and feel" of the application program. "A software program is said to be 'portable' across various platforms if the program can run without modification on any of those platforms." (Specification, page 6, lines 23-24, emphasis added). Before disclosing such means, the specification highlights several drawbacks of prior art attempts at application portability.

For example, Java application programs utilize a platform-dependent application program interface (API), commonly known as the Abstract Windowing Toolkit (AWT), to produce heavyweight software components that are written in native code (i.e., instructions specific to a particular OS). When AWT heavyweight software components are used for displaying images, the "look and feel" of those images may differ depending on the particular OS running the Java application program. In the context of a graphical user interface (GUI), "the 'look and feel' of a GUI refers to such things as the appearance, color and behavior of Buttons, TextFields, Listboxes, menus, etc..." (Specification, page 7, line 4-8). Thus, there are certain limitations within the AWT that prohibit true portability of Java application programs, especially in terms of the look and feel of the application program. *See*, e.g., Specification, page 7, line 24 to page 9, line 10, and page 18, line 10 to page 19, line 23.

In an effort to overcome the platform-dependency of AWT-based programs, Swing was developed as part of the Java Foundation Classes (JFC). An API written using Swing contains no native code, and therefore, can be run on substantially any OS without changing the look and feel of the application. *See*, e.g., Specification, page 9, lines 14-30, and page 19, line 25 to page 20, line 4. Unfortunately, the lightweight software components of Swing cannot completely eliminate the platform-dependency of Java applications that use AWT. Since Swing versions of many AWT components (e.g., containers, such as frames) are unavailable, many programmers have attempted to mix Swing and AWT components within a given API. *See*, e.g., Specification, page 21, lines 1-13. However, straightforward mixing of Swing and AWT components tends to create many problems that programmers have simply learned to accept. *See*, e.g., Specification, page 22, lines 1-28.

One problem encountered when mixing Swing and AWT components within a given API arises when programmers attempt to replace the dual-mode, AWT TextField component with a lightweight Swing counterpart. In AWT, the functionalities of a normal text field and a password-protected text field are both included within the AWT TextField component. However, the normal and password-protected functionalities are not included within a single Swing component, but instead, are separately allocated to the Swing JTextField and JPasswordField components. If a Java application program originally displays an object using the AWT TextField component, the behavior of that program cannot be maintained by simply replacing the AWT TextField component with a single Swing component. *See*, e.g., Specification, page 38, line 17 - page 39, line 5.

In order to utilize Swing components, while maintaining behavioral compatibility with the Java application program, the present inventors recognized that the appropriate Swing component must be dynamically recreated and substituted for the AWT component each time the mode of use of the object changes. This is in direct contrast to the manner in which Swing components are normally created (i.e., during the initial construction of the peer component). *See*, e.g., Specification, page 39, lines 7-8.

Therefore, the presently claimed case provides a system, method and computer-readable storage device for overcoming one of the many problems that occurs when programmers attempt to mix AWT and Swing components within a Java application program. In general, the presently claimed case provides a system of software components – referred to as "AWTSwing" components – that enable an object to be displayed using one of two different Swing software components, which may be dynamically created at runtime depending on a mode of use of the object. In other words, the presently claimed system of software components provides a unique mode-switching capability that allows two Swing software components to alternate as replacements for an AWT software component, depending on the manner in which the object is being used. *See*, e.g., Specification, page 38, line 17 - page 40, line 6.

To provide mode-switching capability, the presently claimed system of software components includes a first proxy component (e.g., JTextFieldProxy 174), a second proxy component (e.g., JPasswordFieldProxy 176) and a peer component (e.g., JTextFieldPeer 172), as shown in Fig. 17. As used herein, the term "software component" may be defined as "any sequence of executable code." (Specification, page 3, line 16). As such, a "proxy component" may be generally defined as a sequence of executable code that, when executed, (i) translates the method calls from an AWT component to the appropriate Swing component(s), and (ii) translates the event calls from a Swing component to the AWT

component (e.g., a frame) containing the Swing component. As used by the proxy component, a "method call" may be defined as a get routine to another software component (i.e., a "graphics resource component") that creates and manipulates a graphical representation of the object to therefore display the object. *See*, e.g., Specification, page 23, lines 16-18, and page 29, lines 7-10.

In present claim 1, the "peer component" is used for selecting either the first proxy component or the second proxy component, depending on a mode of use of the object. In the context of a Text Field box, the "mode of use" may correspond to whether entered text is to be displayed with masked (e.g., "*****") or unmasked (e.g., "hello") text. Once selected by the peer component, the selected proxy component will create a new graphics resource component (e.g., a Swing JTextField or JPasswordField software component) for displaying the object. Since the object is displayed with a Swing component, rather than an AWT component, the appearance of the displayed object will be substantially independent of the operating system running the Java application program. In other words, the properties that define the "look and feel" of the object (e.g., the color, position, text, etc.) will not be affected by the operating system running the Java application program.

As described in more detail below, none of the cited art, either separately or in combination, provides motivation to teach or suggest a system of software components, a computer-readable storage device or a method for selecting, during runtime, a first proxy component or a second proxy component for use in displaying an object, where the selection is dependent on a mode of use of the object and where the appearance of the displayed object is substantially independent of the operating system. Therefore, the teachings of the cited art cannot be used to render the limitations of the presently claimed case unpatentable.

ISSUE 1 ARGUMENTS

Patentability of Group I Claims 1-12

1. **None of the cited art teaches or suggests a system of software components including a first proxy component, a second proxy component and a peer component for displaying an object, where the peer component is configured for selecting, during runtime, either the first proxy component or the second proxy component, depending on a mode of use of the object.**

Independent claim 1 states, in part:

> A system of software components adapted to display an object ... wherein the system of software components comprises: a first proxy component; a second proxy component; and a peer component for selecting either the first proxy component or the second proxy component, depending on a mode of use of the object, wherein the selection can be made during runtime, and wherein the selected proxy component thereafter creates a new graphics resource component for displaying the object, such that the appearance of the displayed object is substantially independent of the operating system.

As noted in the Response to the Office Action Mailed January 15, 2004, neither WinZip nor Java, either separately or in combination, can be used to teach or suggest the presently claimed system of software components. For example, the screen shots provided on pages 3 and 4 of the WinZip reference can only be used to show how various objects (e.g., frames, dialog boxes, buttons, check boxes, text boxes, etc.) may be displayed when the "Mask password" checkbox is toggled between unchecked (page 3) and checked (page 4). More specifically, the screen shots on pages 3 and 4 of the WinZip reference can only be used to show how the Password text box may display masked text ("*****") or unmasked text ("hello"), depending on a mode of use of the Password text box.

However, the screen shots on pages 3 and 4 of the WinZip reference cannot be used to provide teaching or suggestion for the presently claimed system of software components, which as noted above, include a first proxy component, a second proxy component and a peer component. Though the screen shots may show the end product (i.e., the WinZip GUI) obtained by the execution of one or more (inherently included) WinZip software components, the screen shots cannot provide insight into the particular software components (where "software components" are defined as sequences of executable code) that were used to produce the end product. The WinZip reference simply does not and cannot teach or suggest that the presently claimed software components -- i.e., the first proxy component, the second proxy component and the peer component -- are used for displaying the objects shown in the WinZip GUI.

The WinZip reference also fails to provide inherent teaching or suggestion for the presently claimed system of software components. For example, the WinZip reference does not teach or suggest that the functionality shown in the screen shots (i.e., the alternation between masked and unmasked text in the Password text box) could be provided by software components similar to the presently claimed proxy and peer components. In other words, given the functionality disclosed in the screen shots on pages 3 and 4 of the WinZip reference, one skilled in the art could not reasonably conclude that a first

proxy component, a second proxy component and a peer component (or equivalent software components demonstrating the presently claimed functionality) would necessarily be included to provide such functionality.

First of all, the WinZip reference fails to disclose the programming language used to produce the WinZip GUI. The Applicant concedes that one skilled in the art may assume that an object-oriented programming (OOP) language was used to construct the software components needed to produce the WinZip GUI, simply because such languages are typically used in developing GUIs. Thus, for the sake of argument, we may assume that the WinZip screen shots are produced by one or more software components written purely in the Java programming language (a popular OOP language, and therefore, a reasonable assumption). After making such an assumption, one skilled in the art may also assume that a Java peer component (e.g., the heavyweight AWT TextComponentPeer) and a Java graphics resource component (e.g., the heavyweight AWT TextField) are included within the WinZip software components. If this is the case, the Java peer component could invoke the methods of the Java graphics resource component, which would then display the object. However, the appearance of the displayed object would not be independent of the operating system, as in the presently claimed case, because a heavyweight graphics resource component (containing native code) would be used for displaying the object in the WinZip GUI. Furthermore, since the methods for displaying text as either masked or unmasked are both provided within the existing Java graphics resource component (e.g., within the AWT TextField component), the Java peer component would have no need for selecting between a first proxy component and a second proxy component. In fact, if the WinZip software components were written purely in Java, there would be no need for including proxy components that, when selected by the peer component, function to dynamically create new graphics resource components for displaying the object. Thus, if one were to assume that the WinZip software components were written purely in Java, the WinZip software components would not include the first proxy component, second proxy component and peer component, as recited in present claim 1.

In addition, the Java reference cannot be combined with the WinZip reference to overcome the deficiencies therein. For example, the Java reference fails to provide teaching or suggestion for a first proxy component, a second proxy component and a peer component for selecting between the first and second proxy components, depending on a mode of use of an object. Since WinZip and Java each fail to teach or suggest the presently claimed system of software components, the cited art cannot be combined to do so.

Even if one were to make the combination proposed by the Examiner (without sufficient motivation to do so), the proposed combination would still fail to disclose all limitations of present claim 1. Lets assume, for example, that a programmer wishes to replace the above-mentioned Java graphics resource component (e.g., the heavyweight AWT TextField) with a Swing graphics resource component, i.e., a lightweight software component. If the programmer attempted to make such a replacement, he/she would run into several problems.

First of all, the Java graphics resource component may have combined features (e.g., masked and unmasked text capabilities) that simply do not exist within a single Swing graphics resource component. Therefore, the programmer could not directly replace the Java component with an equivalent Swing component (since none exists). Even if the programmer were to replace the Java graphics resource component with multiple Swing graphics resource components (e.g., by modifying the existing Java peer component to invoke the methods of the Swing, rather than the AWT, graphics resource components), the replacement Swing components would not behave properly (e.g., they would not respond to action events, such as mouse clicks or keyboard entries), since the replacement Swing components would not be declared as contained within a Frame of the Java application program. *See*, e.g., Specification, page 22, lines 23-28.

In addition, one skilled in the art would not consider the presently claimed proxy components to be an obvious feature in light of the WinZip and Java references. For example, Swing components are normally created only during the initial construction of the peer component (*See*, e.g., Specification page 39, lines 7-25). Therefore, one skilled in the art would not consider the dynamic creation of Swing components to be an obvious feature. As a consequence, the presently claimed proxy components (which dynamically create the appropriate Swing component during runtime) cannot be considered obvious features in light of the WinZip and Java references.

The arguments provided above clearly show how the WinZip and Java references fail to teach or suggest a first proxy component, a second proxy component and a peer component for selecting either the first or second proxy component, depending on a mode of use of an object. On page 3 of the final Office Action, the Examiner admits that "WinZip and Java ... don't teach using a peer component for selection between two proxy components." Applicants appreciate the Examiner's recognition of any lack of teaching within WinZip and Java for the presently claimed software components.

However, the Examiner suggests that "Fults teaches a system which generates [a user] interface based on a selection by the user (see column 3, lines 27-52) similar to that of WinZip and Java ..." (Final Office Action, page 3). The Examiner suggests that Fults "further teaches taking hints from a user and using the hints to direct the interface generation to an appropriate user interface implementation (see column 3, lines 33-51, figure 2, and the abstract)." (Final Office Action, page 3). Applicants respectfully disagree with the Examiner's alleged evidence of teaching within Fults for the presently claimed peer and proxy components. Reasons supporting why the teachings of Fults fail to mention the presently claimed software components are set forth in more detail below.

For example, pointing out that Fults "teaches taking hints from a user and using the hints to direct the interface generation to an appropriate user interface implementation" does not provide evidence of any teaching within Fults for the system of software components recited in present claim 1. In other words, Fults does not teach or suggest a peer component for displaying an object by selecting either a first or second proxy component, depending on a mode of use of the object.

Any selection between software components that may or may not be disclosed by Fults is not dependent on a mode of use of an object. Instead, Fults creates a generic user interface in the form of attributes and hints (provided by the programmer at the time of code generation) that may be used to produce a user interface, "which visually and behaviorally conforms to the explicit and implicit guidelines of a particular style guide." (*See*, e.g., Fults, column 9, line 55 to column 10, line 26; and column 15, line 62 to column 16, line 16). In other words, Fults discloses that an "application simply defines its user interface using a generic model... [o]nce the application's user interface is described in generic terms, the [invention of Fults] maps each generic UI object to one or more specific UI objects, depending on which specific user interface [i.e., particular style guide] is chosen." (*See*, e.g., Fults, column 16, lines 27-45; column 17, lines 21-25). "For example, an application's UI file might specify that a list of options be presented to the user. Depending on the attributes and 'hints' of the generic object, this might be implemented as a submenu in OpenLook or as a dialog box in OSF/Motif $_{TM}$." (Fults, column 16, lines 45-50). Therefore, Fults discloses a generic user interface, whose objects (e.g., a list of options) may adopt a particular look and feel (e.g., the list may be displayed as a submenu or dialog box), depending on the "hints" provided by the programmer and the particular style guide (e.g., OpenLook or OSF/Motif $_{TM}$) used to display the objects. However, the "hints" and "style guides" disclosed by Fults cannot be considered "modes of use."

For example, "hints" are described by Fults as an "embodiment of human/computer interface criteria for an application, stored digitally. Examples of such criteria follow: 'infrequently used feature,' 'advanced feature,' 'should be displayed as large as possible.'" (Fults, column 6, lines 2-8). Obviously, such "digitally stored criteria" cannot be considered a "mode of use of an object," as taught by the presently claimed case. In addition, "style guides" are described by Fults as a "document intended to impose visual and operational consistency across the set of applications running in a particular environment." (Fults, column 13, lines 17-21). However, "style guides" (like OpenLook or OSF/Motif TM) are not "modes of use of an object," as taught in the presently claimed case.

Therefore, even if Fults were to disclose a peer component for displaying an object by selecting either a first or a second proxy component (which he does not), the selection would not be dependent on a mode of use of the object. As a consequence, the teachings of Fults fail to disclose all limitations of the system of software components recited in present claim 1.

For at least the reasons set forth above, WinZip, Java and Fults each fail to teach or suggest all limitations of present claim 1.

2. **There is no motivation to combine or modify the teachings of the cited art to provide a proxy component configured for displaying an object by selecting, during runtime, either a first or a second proxy component, depending on a mode of use of the object.**

To establish a case of *prima facie* obviousness of a claimed invention, all of the claim limitations must be taught or suggested by the prior art. *In re Royka*, 490 F.2d 981, 180 USPQ 580 (C.C.P.A 1974); MPEP 2143.03. Obviousness can only be established by combining or modifying the teachings of the prior art to produce the claimed invention where there is some teaching, suggestion, or motivation to do so. *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed.Cir. 1988); *In re Jones*, 958 F.2d 347, 21 USPQ2d 1941 (Fed. Cir. 1992); MPEP 2143.01.

As noted above, WinZip, Java and Fults each fail to separately provide teaching or suggestion for the presently claimed peer and proxy components. In fact, the Examiner admits that "WinZip and Java... don't teach using a peer component for selection between two proxy components." (Final Office Action, page 3). However, the Examiner suggests that "[i]t would have been obvious to one of ordinary skill in the art, having the teachings of WinZip, Java, and Fults, before him at the time the invention was made to modify the password display system of WinZip and Java to be implemented with a peer component that

directs the interface development to the appropriate user interface implementation. One would have been motivated to make such a combination because WinZip has a switch of display characteristics very similar to that of Fults, where upon a user selection, an element is displayed using a different API component." (Final Office Action, page 3). The Applicant's respectfully disagrees with the Examiner's motivation for combining the teachings of WinZip and Fults, and further asserts that neither WinZip nor Fults provide motivation to make such a combination.

As noted above, the screen shots of WinZip can only be used to show how various objects (e.g., frames, dialog boxes, buttons, check boxes, text boxes, etc.) may be displayed in a graphical user interface (GUI) when the "Mask password" checkbox is toggled between unchecked (page 3) and checked (page 4). Given the screen shots of WinZip, however, it is simply impossible to determine the software components used to realize the functionality shown therein. For example, the Applicants have theorized that the Java programming language could have been used to produce the WinZip GUI. According to such theory, the "switch of display characteristics" of WinZip (e.g., the switch between masked and unmasked text) may have been provided by the AWT TextField component. However, the screens shots of WinZip do not suggest, nor do they provide motivation for, implementing the "switch of display characteristics" with two different API components. Therefore, contrary to the Examiner's suggestion, the "switch of display characteristics" of WinZip are NOT necessarily "similar to that of Fults, where upon a user selection, an element is displayed using a different API component." As a consequence, the Examiner's motivation for combining the teachings of WinZip and Fults is considered insufficient.

Furthermore, WinZip and Fults cannot be combined to provide the presently claimed peer and proxy components, since neither WinZip nor Fults suggest the desirability of doing so. The mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination. *In re Mills*, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990); MPEP 2143.01. The invention of Fults provides a generic user interface, including attributes and hints, which can be interpreted during runtime to produce a user interface that conforms to the look and feel of a particular style guide. If there is any selection within Fults between two proxy components, such selection is based on the particular style guide used to display the GUI elements. Fults provides absolutely no desirability for selecting between two proxy components, depending on a mode of use of a displayed object. Therefore, Fults provides no desirability to be combined with the teachings of WinZip or modified to base its selection on a mode of use of a displayed object.

For at least the reasons set forth above, WinZip and Fults cannot be combined or modified to teach or suggest all limitations of present claim 1.

3. There is no motivation to combine the text display system of WinZip with the system independence of Java, as suggested by the Examiner.

Independent claim 1 teaches, that after the appropriate proxy component is selected, "the selected proxy component thereafter creates a new graphics resource component for displaying the object, such that the appearance of the displayed object is substantially independent of the operating system." In this manner, the presently claimed case provides a unique mode-switching capability that may allow two Swing software components (e.g., the JTextField and JPasswordField components) to alternate as replacements for an AWT software component (e.g., the TextField component), depending on the mode of use of an object (e.g., a text box). Since Swing software components do not contain native code, the appearance of the displayed object will be substantially independent of the operating system running the application program.

Conversely, the WinZip and Java references provide absolutely no teaching or suggestion for combining AWT and Swing software components within a Java application program that, when executed, displays an object in a manner that is independent of the operating system running the Java application. However, on page 3 of the Office Action mailed January 15, 2004, the Examiner suggests that it would have been obvious to one skilled in the art "to modify the text display system of WinZip to include the system independence of [the] Java [reference]." Though arguments were provided in the Response to that Office Action, the Examiner suggestions were maintained in the final Office Action mailed June 30, 2004. As previously noted, the Applicant respectfully disagrees with the Examiner's position that the text display system of WinZip can be modified to include the system independence of the Java reference. Reasons as to why the WinZip and Java references cannot be combined are provided below.

As noted in the Response to the Office Action mailed January 15, 2004, the Java reference is merely an overview, or a brief description, of the Swing JTextField and JPasswordField components. Like the WinZip reference, the Java reference fails to provide any teaching or suggestion for combining AWT and Swing software components within a Java application program. Neither reference indicates that AWT and Swing software components can be combined in a manner that provides the presently claimed system of software components or the functionality thereof. Thus, the cited art fails to provide

motivation for modifying the text display of the WinZip reference (which could reasonably include AWT components) to include the system independence of the Java references (which could be provided by Swing components). The Applicant asserts that the teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, not in the Applicant's disclosure. *In re Vaech*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991); emphasis added, MPEP 2143.

Without the benefit of Applicant's disclosure, one skilled in the art would not be motivated to combine the text display system of WinZip with the system independence of Java given the teachings of WinZip, Java and the ordinary knowledge of one skilled in the art.

In addition, the Applicants disagree with the Examiner's motivation for combining the WinZip and Java references, which states that "[o]ne would have been motivated to make such a combination because this would allow people on different platforms to use similar and familiar interfaces." (*See*, e.g., Office Action, page 3; Final Office Action, page 3). Since the Java reference, and more specifically, the concept of Swing, already allows "people on different platforms to use similar and familiar interfaces", one skilled in the art would not be motivated to combine the system independence of Java with the WinZip application program, which is most likely dependent on the operating system running the WinZip application program. Thus, Applicant's assert that the Examiner's motivation for combining the teachings of WinZip and Java is insufficient.

4. The Examiner has failed to adequately support and/or establish a *prima facie* ground of obviousness.

To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all claim limitations. MPEP § 2143. None of these three criteria have been met by the Examiner in the present case. First of all, no suggestion or motivation to modify the cited references can be found within the cited references to teach or suggest the aforementioned limitations of present claims 1, as explained above in Arguments 2 and 3. The criterion of a reasonable expectation of success cannot be met if no teaching, suggestion or motivation exists, because there is then nothing at which to be successful. Finally, none of the cited art, either alone or in combination,

teaches all of the limitations of claim 1, as explained above in Argument 1. The third criterion recited above has therefore also not been met, and a *prima facie* case of obviousness has not been established.

Conclusion

As explained in Arguments 1-4 above, at least some limitations of claim 1 and at least some limitations of claims 2-12, are not taught or suggested by the cited art. Furthermore, there is no teaching, suggestion or motivation to modify the cited art to teach the limitations of these claims. For at least the reasons set forth above, claims 1-12 are patentably distinct over the cited art, and the rejection of Group I claims 1-12 under 35 U.S.C. § 103(a) is asserted to be erroneous.

ISSUE 2 ARGUMENTS

Patentability of Group II Claims 13-25

Because claims 13-25 of Group II include limitations similar to those included within claim 1 of Group I, the arguments presented above for patentability of claim 1 apply equally to claims 13-25, and are herein incorporated by reference. In addition to the arguments presented above with respect to claim 1, arguments are provided below to further establish the patentability of the current claims under 35 U.S.C. §103(a).

1. **None of the cited art provides motivation to teach or suggest an application program for activating a first proxy component to dynamically generate a first graphical representation of an object, and upon detecting a change in the mode of use of the object, deactivating the first proxy component and then activating a second proxy component to dynamically generate a second graphical representation of an object, where the first and second graphical representations (one being distinct from the other) are substantially independent of an operating system.**

Independent claim 25 recites, in part:

> A computer-readable storage device, comprising... an application program running under the operating system... wherein the application program is adapted for: activating a first proxy component to dynamically generate a first graphical representation of the object during runtime that is substantially independent of the operating system; monitoring the mode of use of the object; and upon detecting a change in the mode of use of the object, deactivating the first proxy component and activating a second proxy component to dynamically generate a second graphical representation of the object during runtime, wherein the second graphical representation is substantially independent of the operating system and distinct from the first graphical representation.

Independent claim 13 recites a similar limitation. In addition, claims 13 and 25 recite limitations similar to those recited in present claim 1, in that they disclose various method steps that may be performed upon execution of the system of software components recited in present claim 1.

On page 7 of the Office Action, the Examiner admits that the WinZip reference fails to provide any teaching or suggestion for the appearance of the displayed object (i.e., the graphical representation of the object) being independent of the operating system. The Applicant agrees. However, the Applicant does not agree that the teachings of the WinZip and Java references can be combined to render the above limitation obvious, as is also suggested by the Examiner (e.g., on page 9 of the Final Office Action). As noted above in Argument 3 of the Group I claims, neither reference provides sufficient motivation that would enable one skilled in the art to make the proposed combination. The Examiner cannot rely on the Applicant's disclosure for providing such motivation.

On pages 8 and 9 of the final Office Action, the Examiner admits that the WinZip and Java references do not teach using a peer component for selection between two proxy components. The Applicant agrees. However, the Applicant does not agree that the teachings of the WinZip, Java and Fults references can be combined to render the above limitation obvious, as is also suggested by the Examiner (e.g., on pages 9 and 10 of the Final Office Action). As noted above in Argument 1 of the Group I claims, Fults does not disclose the above limitation, and therefore, cannot be combined with WinZip and Java to overcome the deficiencies therein. As noted above in Argument 2 of the Group I claims, none of the references provides sufficient motivation that would enable one skilled in the art to make the proposed combination. The Examiner cannot rely on the Applicant's disclosure for providing such motivation.

Since the cited art cannot be combined to provide teaching or suggestion for the peer and proxy components recited in present claim 1, the cited art cannot be relied upon to disclose the various method steps (recited in claims 13 and 25) performed upon execution of the presently claimed peer and proxy components. In other words, the cited art does not teach, nor can they be combined or modified to teach, an application program adapted for activating a first proxy component to dynamically generate a first graphical representation of an object, and upon detecting a change in the mode of use of the object, deactivating the first proxy component and then activating a second proxy component to dynamically generate a second graphical representation of an object, where the first and second graphical representations (one being distinct from the other) are substantially independent of an operating system. As described in more detail below, the cited art simply fails to disclose all limitations of present claims 13 and 15.

For example, though the screen shots of WinZip show that a text box can display either masked or unmasked text, the screen shots of WinZip cannot be relied upon to teach the presently claimed steps of activating a first proxy component (e.g., to display an object), monitoring a mode of use of the object, and upon detecting a change in the mode of use of the object, deactivating the first proxy component and activating a second proxy component (e.g., to display the object in a different manner). Since the screen shots of WinZip fail to disclose the existence of different software components, the screen shots provide absolutely no teaching, suggestion or motivation for activating/deactivating different software components. The screen shots of WinZip also lack teaching for monitoring a mode of use of the object displayed by the proxy components. Instead, the manner in which text is displayed in the screen shots of WinZip depends on whether or not a "Mask password" checkbox is currently unchecked (page 3, WinZip) or checked (page 4). Therefore, if any monitoring is inherently disclosed within the screen shots of WinZip, such monitoring may only be used to detect a mode of use of the "Mask password" checkbox. By monitoring the mode of use of the "Mask password" checkbox, the screen shots of WinZip fail to disclose the presently claimed step of monitoring a mode of use of the object (e.g., the text box) displayed by the proxy components. As a consequence, the screen shots of WinZip fail to disclose all limitations of present claims 13 and 25.

In addition, Fults and Java cannot be combined with WinZip to overcome the deficiencies therein. The Java reference, which provides a brief overview of the Swing JTextField and JPasswordField components, obviously fails to disclose the presently claimed steps of activating, monitoring and deactivating. Thus, the combination of WinZip and Java fails to disclose all limitations of present claims 13 and 25. Though Fults mentions that "the user may switch specific user interfaces (e.g. Motif to Openlook) at any time" (Fults, column 16, lines 16-17), the switching of specific user interfaces (otherwise referred to as "style guides") does not provide teaching or suggestion for the presently claimed steps of activating, monitoring and deactivating. As noted above, the "style guides" disclosed by Fults cannot be considered a "mode of use of an object", as taught by the presently claimed case. Thus, Fults fails to provide teaching or suggestion for monitoring a mode of use of an object, and upon detecting a change in the mode of use of the object, deactivating a first proxy component and activating a second proxy component, as recited in present claims 13 and 21. As a consequence, Fults fails to disclose all limitations of present claims 13 and 25, and therefore, cannot be combined with WinZip and/or Java to do so.

2. **The Examiner has failed to adequately support and/or establish a *prima facie* ground of obviousness.**

To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all claim limitations. MPEP § 2143. None of these three criteria have been met by the Examiner in the present case. First of all, no suggestion or motivation to modify the cited references can be found within the cited references to teach or suggest the aforementioned limitation of claims 13 and 25, as explained in the above Argument. The criterion of a reasonable expectation of success cannot be met if no teaching, suggestion or motivation exists, because there is then nothing at which to be successful. Finally, none of the cited art, either alone or in combination, teaches all of the limitations of claims 13 and 25, as explained in the above Argument. The third criterion recited above has therefore also not been met, and a *prima facie* case of obviousness has not been established.

Conclusion

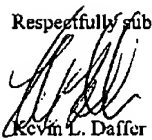
As explained in Arguments 1-2 above, at least some limitations of claims 13 and 25, as well as at least some limitations of claims 14-24, are not taught or suggested by the cited art. Furthermore, there is no teaching, suggestion or motivation to modify the cited art to teach the limitations of these claims. For at least the reasons set forth above, claims 13-25 are patentably distinct over the cited art, and the rejection of Group II claims 13-25 under 35 U.S.C. §103(a) is asserted to be erroneous.

IX. CONCLUSION

For the foregoing reasons, it is submitted that the Examiner's rejection of claims 1-6, 8-13 and 15-19 was erroneous, and reversal of the Examiner's decision is respectfully requested.

The Commissioner is hereby authorized to charge the required fee(s) to deposit account number 50-3268/5468-07600.

Respectfully submitted,



Kevin L. Daffer
Reg. No. 34,146
Attorney for Appellant

Daffer McDaniel, L.L.P
P.O. Box 684908
Austin, TX 78768-4908
Date: November 23, 2004
JMP